Intermediate Python Programming Course

Course Overview
This course is designed for individuals who have a basic understanding of Python and want to deepen their knowledge and skills. The course will cover advanced topics, best practices, and real-world applications of Python programming. By the end of this course, participants will be able to write more efficient, maintainable, and scalable Python code.

Course Structure
The course is divided into four modules, each focusing on different aspects of intermediate Python programming.

Module 1: Advanced Data Structures

Lesson 1.1: Lists and Tuples
- Understanding list comprehensions
- Using tuples for immutable data

Code Example:
```python
# List comprehension example
squares = [x**2 for x in range(10)]
print(squares)

# Tuple usage
coordinates = (10.0, 20.0)
print(f"X: {coordinates[0]}, Y: {coordinates[1]}")
```

Lesson 1.2: Dictionaries and Sets
- Advanced dictionary methods
- Set operations and their applications

Code Example:
```python
# Dictionary example
student_scores = {'Alice': 85, 'Bob': 90, 'Charlie': 78}
average_score = sum(student_scores.values()) / len(student_scores)
print(f"Average Score: {average_score}")
```

# Set operations
set_a = {1, 2, 3}
set_b = {3, 4, 5}
intersection = set_a & set_b
print(f"Intersection: {intersection}")
```

#### Lesson 1.3: Collections Module
- Using namedtuples, defaultdict, and Counter

**Code Example:**
```python
from collections import namedtuple, defaultdict, Counter

# Namedtuple example
Point = namedtuple('Point', ['x', 'y'])
p = Point(10, 20)
print(p)

# Defaultdict example
dd = defaultdict(int)
dd['a'] += 1
print(dd)

# Counter example
word_count = Counter(['apple', 'banana', 'apple', 'orange'])
print(word_count)
```

Module 2: Object-Oriented Programming (OOP)

Lesson 2.1: Classes and Objects
- Defining classes and creating objects
- Understanding self and __init__

Code Example:
```python
class Dog:
    def __init__(self, name):
        self.name = name

    def bark(self):
        return f"{self.name} says Woof!"

my_dog = Dog("Buddy")
print(my_dog.bark())
```

Lesson 2.2: Inheritance and Polymorphism
- Creating subclasses and overriding methods

Code Example:
```python
class Animal:
    def speak(self):
        return "Animal speaks"

class Cat(Animal):
    def speak(self):
        return "Meow"

my_cat = Cat()
print(my_cat.speak())
```

Lesson 2.3: Magic Methods
- Understanding dunder methods for operator overloading

Code Example:
```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

v1 = Vector(2, 3)
v2 = Vector(5, 7)
result = v1 + v2
print(f"Result Vector: ({result.x}, {result.y})")

```

Module 3: Functional Programming

Lesson 3.1: Lambda Functions
- Using lambda functions for concise code

Code Example:
```python
# Lambda function example
add = lambda x, y: x + y
print(add(5, 3))
```

Lesson 3.2: Map, Filter, and Reduce
- Applying functional programming concepts

Code Example:
```python
from functools import reduce

# Map example
numbers = [1, 2, 3, 4]
squared = list(map(lambda x: x**2, numbers))
print(squared)

# Filter example
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(evens)

# Reduce example
product = reduce(lambda x, y: x * y, numbers)
print(product)
```

Lesson 3.3: Decorators
- Creating and using decorators

Code Example:
```python

```
def decorator_function(original_function):
    def wrapper_function():
        print("Wrapper executed before {}".format(original_function.__name__))
        return original_function()
    return wrapper_function

@decorator_function
def display():
    return "Display function executed"

print(display())
```

Module 4: Working with External Libraries and APIs

Lesson 4.1: Introduction to Libraries
- Installing and using third-party libraries (e.g., requests, NumPy)

Code Example:
```python
import requests

response = requests.get('https://api.github.com')
print(response.json())
```

Lesson 4.2: Data Manipulation with Pandas
- Using Pandas for data analysis

Code Example:
```python
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Score': [85, 90, 78]}
df = pd.DataFrame(data)
print(df)

# Calculate average score
average_score = df['Score'].mean()
print(f"Average Score: {average_score}")
```

Lesson 4.3: Building a Simple API with Flask
- Creating a RESTful API using Flask

Code Example:
```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/api/greet/<name>', methods=['GET'])
def greet(name):
    return jsonify(message=f"Hello, {name}!")

if __name__ == '__main__':
    app.run(debug=True)
```

Conclusion
By completing this intermediate Python programming course, participants will have a solid understanding of advanced data structures, object-oriented programming, functional programming, and working with external libraries and APIs. This knowledge will empower them to tackle more complex programming challenges and build robust applications using Python.